CS-303 Software Engineering

GROUP 15

Disaster Alert System

Step-by-Step Documentation

Introduction

The Disaster Alert System is designed to provide real-time alerts to users in affected regions. This system ensures timely communication of disaster warnings to mitigate risks and improve response times. The key focus areas of this project include:

- **Usability:** A user-friendly interface for sending and receiving alerts.
- **Reliability:** A robust system that can handle high traffic during emergencies.
- **Safety:** Accurate alerts with minimal false positives or negatives.

This document provides a comprehensive guide to the system's setup, functionality, architecture, security measures, and future improvements.

---

1. Setting Up Email for Sending Alerts

1.1 Creating a Gmail Account (If Not Already Available)

1. Go to Gmail and sign up for a new account if you don't have one.
2. Choose a professional email ID for the alert system.

1.2 Enabling 2-Step Verification

1. Sign in to your Gmail account.
2. Go to Google Account Security.
3. Under "Signing in to Google," click **2-Step Verification** and follow the instructions to enable it.

1.3 Generating an App Password

1. Go to Google App Passwords.
2. Select **Mail** as the app and **Other (Custom Name)** for the device.
3. Generate the password and store it safely for later use in the code.

---

2. Setting Up the Database

2.1 Installing SQLite (If Not Installed)

SQLite comes pre-installed in Python. If needed, install it using:

pip install sqlite3

2.2 Creating the Users Database

A database file users.dbwill be created to store users and their locations.

2.2.1 Creating the **users**Table

The table will have the following columns:

- id(Primary Key, Auto Increment)
- name(User Name)
- location(User Location)
- email(User Email)

2.2.2 Populating the Database

We will insert test users as follows:

2.2.2.1 a1to z1from loc_1
2.2.2.2 a2to z2from loc_2
2.2.2.3 ...
2.2.2.4 a10to z10from loc_10

The respective emails will follow the pattern:

2.2.2.5 a1_loc1@gmail.com
2.2.2.6 b1_loc1@gmail.com
2.2.2.7 ...
2.2.2.8 z10_loc10@gmail.com

This will be done in the Python script using sqlite3.

---

3. Implementing the Alert System

3.1 Required Libraries

Install the required dependencies using:

```
pip install tkinter smtplib sqlite3
```

3.2 Understanding the Code Structure

The script performs the following functions:

1. **Database Setup:** Creates the database and inserts user records.
2. **GUI Creation:** Uses Tkinter to build a user-friendly interface.
3. **Email Handling:** Connects to Gmail's SMTP server and sends alerts.
4. **User Selection:** Fetches relevant user emails based on location.
5. **Logging Alerts:** Stores sent alerts for record-keeping.

3.3 Setting Up the Database in Python

The database is created using the following Python code:

```
import sqlite3

conn = sqlite3.connect('users.db')
cursor = conn.cursor()
cursor.execute('''CREATE TABLE IF NOT EXISTS users (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                name TEXT,
                location TEXT,
                email TEXT)''')
conn.commit()
```

This ensures that if the database doesn't exist, it is created with the required structure.

3.4 Populating the Database with Users

```
locations = [f'loc_{i}' for i in range(1, 11)]
users = [(f'{chr(97+j)}{i}', loc,
        f'{chr(97+j)}{i}_{loc}@gmail.com') for i, loc in
        enumerate(locations, 1) for j in range(26)]

cursor.executemany("INSERT INTO users (name, location, email) VALUES (?, ?, ?)",
users) conn.commit()
conn.close()
```

This dynamically generates user data with names and emails in the expected format.

3.5 Sending Email Alerts

The script connects to Gmail's SMTP server using:

```
with smtplib.SMTP_SSL('smtp.gmail.com', 465) as server:
    server.login(sender_email, app_password)
```

Emails are sent using the ***send_email*** function:

```
def send_email(recipient, subject, message):
    msg = f"Subject: {subject}\n\n{message}"
    server.sendmail(sender_email, recipient,
    msg)
```

3.6 Implementing the GUI

The system features a **graphical user interface (GUI)** built with Tkinter. Key components include:

- **Dropdown Menu**: Allows users to select a disaster-affected location.
- **Text Area**: Enables users to enter a custom disaster alert message.
- **Buttons**:
    - **Send Alert**: Dispatches emails to users in the selected region.
    - **View Logs**: Displays a history of sent alerts for verification.

Example GUI Implementation:

```
import tkinter as tk

def send_alert():
    location = location_var.get()
    alert_message = alert_text.get("1.0", tk.END)
    # Fetch users from the database and send alerts

tk.Label(root, text="Select Location:").pack()
location_var = tk.StringVar()
tk.OptionMenu(root, location_var, *locations).pack()
tk.Button(root, text="Send Alert",
command=send_alert).pack()
```

---

4. Running the Disaster Alert System

4.1 Executing the Script

Run the script using:

python disaster_alert.py

4.2 Sending Alerts

1. Select a location from the dropdown.
2. Enter an alert message.
3. Click "Send Alert."

4.3 Viewing Sent Alerts

Click "View Alerts" to check past logs.

---

5 Troubleshooting

5.1 Common Issues & Fixes

- **SMTP Authentication Error**: Ensure the app password is correct.
- **No Users in Location**: Verify that the database contains user data.
- **GUI Not Showing**: Ensure Tkinter is installed correctly.

---

6 Enhancements & Future Work

To further improve the system, the following features can be integrated:

6.1 Real-Time Alert Tracking

- Maintain a database table (alerts) to log all sent messages.
- Develop an **admin dashboard** for monitoring recent alerts.
- Implement an **API endpoint** for retrieving past alerts.

6.2 SMS Notifications

- Integrate with **Twilio API** to send SMS alerts.
- Example command to install Twilio

6.3 Google Maps Integration

- Use **Google Maps API** to visually mark affected areas.
- Display disaster-affected regions dynamically based on location selection.
- Example using folium

---

7. Security Considerations

To ensure secure operation and protect user data, the following measures are implemented:

- **Environment Variables**: Avoid hardcoding sensitive credentials (use .env files).
- **SMTP Authentication Security**: Use **App Passwords** instead of regular passwords for sending emails.
- **Data Encryption**: Encrypt stored user emails to prevent unauthorized access.
- **Access Control**: Restrict access based on user roles (admin vs. regular users).

Conclusion

The **Disaster Alert System** is an essential tool for emergency response teams, ensuring timely alerts are sent to users in disaster-prone areas. By improving **scalability, security, and real-time tracking**, the system can evolve into a **comprehensive disaster management solution**.
This document provides a complete guide to setting up, coding, running, and troubleshooting the Disaster Alert System. It explains the logic behind the database setup, email handling, GUI creation, and alert-sending mechanisms in detail.